Distributed Particle Swarm Optimization for Limited Time Adaptation in Autonomous Robots

Ezequiel Di Mario and Alcherio Martinoli

Abstract Evaluative techniques offer a tremendous potential for on-line controller design. However, when the optimization space is large and the performance metric is noisy, the time needed to properly evaluate candidate solutions becomes prohibitively large and, as a consequence, the overall adaptation process becomes extremely time consuming. Distributing the adaptation process reduces the required time and increases robustness to failure of individual agents. In this paper, we analyze the role of the four algorithmic parameters that determine the total evaluation time in a distributed implementation of a Particle Swarm Optimization algorithm. For a multi-robot obstacle avoidance case study, we explore in simulation the lower boundaries of these parameters with the goal of reducing the total evaluation time so that it is feasible to implement the adaptation process within a limited amount of time determined by the robots' energy autonomy. We show that each parameter has a different impact on the final fitness and propose some guidelines for choosing these parameters for real robot implementations.

1 Introduction

Human design of high-performing robotic controllers is not a trivial task for a number of reasons. In the first place, even the simplest of modern robots have a large number of sensors and actuators, which implies a large number of control parameters to optimize. Secondly, real systems often present discontinuities and nonlinearities, making it difficult to apply well-understood linear control techniques. Finally, when porting the designed controller to real robots there might be an unexpected

Ezequiel Di Mario and Alcherio Martinoli
Distributed Intelligent Systems and Algorithms Laboratory, School of Architecture, Civil and Environmental Engineering, École Polytechnique Fédérale de Lausanne.
1015 Lausanne, Switzerland
e-mail: ezequiel.dimario@epfl.ch, alcherio.martinoli@epfl.ch.

performance drop due to a number of factors such as imperfections in fabrication, changes in the environment, or modeling inaccuracies.

Machine-learning techniques are an alternative to human-guided design that can address some of the previously mentioned challenges. They can automatically synthesize robotic controllers in large search spaces, coping with discontinuities and nonlinearities, and find innovative solutions not foreseen by human designers by working with a pool of potentially very diverse candidate solutions. Furthermore, the learning process can be implemented fully on-board, enabling automatic adaptation to the underlying hardware and environment.

However, the main drawback of working with a pool of candidate solutions is the amount of time needed to evaluate all candidates, which is substantially larger than that required to generate them. Moreover, due to several sources of uncertainty, such as sensor noise, manufacturing tolerances, or lack of strict coordination in multi-robot settings, it may be necessary to re-evaluate some solutions to gather sufficient statistics for meaningful adaptation. Because of these two reasons, the adaptation process is considered an expensive optimization problem.

Implementing the adaptation process in a distributed fashion brings two distinct advantages. Firstly, it reduces the required evaluation time through parallelization. Secondly, it increases robustness by avoiding a critical point of failure, which is of particular interest in real robot implementations.

Thus, the goal of this paper is to analyze how different algorithmic parameters in a distributed implementation affect the total evaluation time and resulting fitness. We aim to reduce the total evaluation time such that it is feasible to implement the adaptation process within the limits of the robots' energy autonomy without renouncing the benefits of a population-based learning algorithm.

2 Related Work

Particle Swarm Optimization (PSO) is a relatively new metaheuristic originally introduced by Kennedy and Eberhart [8]. PSO is inspired by the movement of flocks of birds and schools of fish, and represents a set of candidate solutions as a swarm of particles moving in a multi-dimensional space. Particles can recall at which position of the search space they obtained their best performance and also the position of the best performing particle in a pre-established neighborhood.

Because of its simplicity and versatility, PSO has been used in a wide range of applications such as antenna design, communication networks, finance, power systems, and scheduling. Within the robotics domain, popular topics are robotic search, path planning, and odor source localization [13].

PSO is well suited for distributed/decentralized implementations due to its distinct individual and social components and the use of the neighborhood concept. Most of the work on distributed implementations has been focused on benchmark functions running on computational clusters [1,3,16]. Implementations with mobile robots are mostly applied to odor source localization [9,17], and robotic search [6],

where, as opposed to optimizing a set of control parameters for the task at hand, the particles' position is usually directly matched to the robots' position in the arena. Thus, the search is conducted in two dimensions and with few or even only one local extrema. For these reasons, even though robotic search is a challenging practical task, it does not represent a complex optimization problem.

An example of a more challenging on-line optimization problem is the work of Floreano and Mondada [5], who used Genetic Algorithms to optimize the weights of an artificial neural network controller. The task was to navigate a path and avoid obstacles with a tethered mobile robot. Even though the population manager and other resource-intensive tasks were carried out on a dedicated off-board computer, this study was still able to show the advantages of evaluation with hardware in the loop. For example, the evolved direction of motion was a result of the interplay between the robot morphology (higher density of proximity sensors facing forward) and the environment in which it was deployed. It is worth noting that the experiment required 67 hours of total evaluation time, and it would require the same time to recreate it nowadays since the limit was not imposed by computational capabilities but rather by the wall-clock time needed to gather enough information on the quality of the candidate solutions.

Most of the research on optimization in noisy environments has focused on evolutionary algorithms [7]. The performance of PSO under noise has not been studied so extensively. Parsopoulos and Vrahatis showed that standard PSO was able to cope with noisy and continuously changing environments, and even suggested that noise may help to avoid local minima [12]. Pan et al. [11] proposed a PSO variation based on statistical tests to select particles, but was only applied to benchmark functions with added gaussian noise.

Pugh et al. showed that PSO could outperform Genetic Algorithms on benchmark functions and for certain scenarios of limited-time learning under the presence of noise [14, 15]. Pugh also showed that PSO can perform satisfactorily with low population sizes, a result that is of particular interest for multi-robot implementations because a smaller number of robots can be used while leaving the optimization process robust to connectivity issues between the robots.

3 Materials and Methods

This paper is focused on a case study of obstacle avoidance, a basic behavior in robotics. Robots navigate autonomously in a square arena of 1 m^2 in which walls and other robots are the only obstacles. We use the same metric of performance as Floreano and Mondada [5], which consists of three factors, all normalized to the interval [0, 1] (Eq. 1).

$$F = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i) \tag{1}$$

V is the average wheel speed, Δv the wheel speed difference, and i the proximity sensor activation value of the most active sensor. Each factor is calculated at each

time step and then averaged for the total number of time steps in the evaluation period. This function rewards robots that move quickly, turn as little as possible, and spend as little time as possible near obstacles.

We chose the obstacle avoidance task because it is scalable in the number of robots, requires basic sensors and actuators that are available in most mobile robots, and the chosen performance metric can be fully evaluated with on-board resources. Thus, it can serve as a benchmark for testing distributed learning algorithms with real robots in the same way that standard benchmark functions are used in numerical optimization.

Our experimental platform is the Khepera III mobile robot, a differential wheeled vehicle with a diameter of 12 cm. The Khepera III is equipped with nine infra-red sensors as well as five ultrasound sensors for short and medium range obstacle detection. Our experiments were carried out in simulation using Webots [10], a realistic, physics-based, submicroscopic simulator that models dynamical effects such as friction and inertia. In this context, by submicroscopic we mean that it provides a higher level of detail than usual microscopic models, faithfully reproducing intra-robot modules (e.g., individual sensors and actuators).

The controller architecture is an artificial neural network of two units with sigmoidal activation functions, and a single output per unit to control the two motors. Each neuron has 12 input connections: the 9 infrared sensors, a connection to a constant bias speed, a recurrent connection from its own output, and a lateral connection from the other neuron's output, resulting in 24 weight parameters in total.

The adaptation algorithm is the noise-resistant variation of PSO introduced by Pugh et al. [15], which operates by re-evaluating personal best positions and aggregating them with the previous evaluations (in our case a regular average performed at each iteration of the algorithm). The movement of particle i in dimension j depends on three components: the velocity at the previous step weighted by an inertia coefficient w, a randomized attraction to its personal best $x^*_{i,j}$ weighted by w_p, and a randomized attraction to the neighborhood's best $x^*_{i',j}$ weighted by w_n (Eq. 2). $rand()$ is a random number drawn from a uniform distribution between 0 and 1.

$$v_{i,j} = w \cdot v_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \qquad (2)$$

The neighborhood presents a ring topology with one neighbor on each side. Particles' positions and velocities are initialized randomly with a uniform distribution in the [-20, 20] interval, and their maximum velocity is also limited to that interval. The robots' pose (position and orientation in the arena) is randomized at the beginning of each evaluation. At the end of each optimization run, the best solution is tested with 40 evaluations of 30 s, and the final performance is the average of these final evaluations.

The total evaluation time for PSO depends on four factors: population size (N_p), individual candidate evaluation time (t_e), number of iterations of the algorithm (N_i), and number of re-evaluations of the personal best position associated with each candidate solution within the same iteration (N_{re}), as shown in Eq. 3.

$$t_{tot} = t_e \cdot N_p \cdot N_i \cdot (N_{re} + 1) \tag{3}$$

If we assume that there is a fixed upper limit for the total evaluation time, an increase in any of the parameters would result in a proportional decrease in the rest.

In a parallelized or distributed implementation, fitness evaluations are distributed among N_{rob} robots, and the wall-clock time t_{wc} required to evaluate candidate solutions is reduced (Eq. 4).

$$t_{wc} = t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot N_i \cdot (N_{re} + 1) \tag{4}$$

It is worth noting that the number of robots is not necessarily the same as the population size. In fact, the choice of the population size depends on the dimension of the search space and the complexity of the task, while the choice of number of robots is based on more experimental considerations (e.g., availability of robots, targeted number of robots needed for a specific mission in a given environment). Thus, a robot may have several particles to evaluate within the same candidate solution pool as opposed to only one.

A full optimization of the algorithmic parameters to minimize the total evaluation time would be a computationally very expensive problem, due to the large number of candidate configurations, the combination of continuous and discrete parameters, the large variation between runs, and the possible existence of local optima. Thus, our approach is to analyze each parameter individually, taking into account its impact on the final performance as compared to a full-time adaptation baseline.

Our baseline set of parameters, based on the work of Pugh et al [14], is shown in Table 1. This set of parameters amounts to a total evaluation time of approximately 417 hours if carried out on a single robot, what we refer to as full-time adaptation.

To complement our robotic case study and add more generality, we also perform runs on four traditional mono and multi-modal benchmark functions without noise: the sphere, Rosenbrock's, Rastrigin's, and Griewank's, defined in Eq. 5. The baseline parameters for the algorithm ran on benchmark functions are also shown in Table 1.

$$\begin{aligned}
f_1(\mathbf{x}) &= \sum_{i=1}^{D} x_i^2 \\
f_2(\mathbf{x}) &= \sum_{i=1}^{D-1} [(1 - x_i^2) + 100(x_{i+1} - x_i^2)^2] \\
f_3(\mathbf{x}) &= 10D + \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i)] \\
f_4(\mathbf{x}) &= 1 + \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}})
\end{aligned} \tag{5}$$

Table 1 Algorithmic parameter values

Parameter	Obstacle Avoidance	Benchmark functions
Population size N_p	100	100
Iterations N_i	50	500
Evaluation span t_e	150 s	-
Re-evaluations N_{re}	1	0
Personal weight w_p	2.0	2.0
Neighborhood weight w_n	2.0	2.0
Dimension D	24	24
Inertia w	0.8	0.6
V_{max}	20	5.12

4 Results and Discussion

The results of this paper are presented as follows: Section 4.1 introduces the discussion with a comparison of the fitness as a function of the total evaluation time. In Sections 4.2 to 4.5 we analyze each of the four previously mentioned algorithmic parameters and propose guidelines for setting their values. Finally, in Section 4.6 we apply the proposed guidelines to reduce the total evaluation time and compare the results with the full-time adaptation.

4.1 Parameter comparison

In the first place, we started from the total evaluation time baseline of 417 h and reduced N_p, N_i, and t_e individually to 5, 5, and 5 s respectively, while keeping the other two parameters at their baseline values, plotting the three curves in the same graph for better comparisons. We performed 100 independent runs for each set of parameter values and with 1, 2, 4, and 8 robots. When multiple robots were considered, all of them were learning in parallel. All runs were performed in a 1x1 m arena unless noted otherwise. The resulting fitness can be observed in Figure 1. In all cases, reducing the evaluation span t_e has the least impact on the resulting fitness, followed by N_p and N_i. When comparing the same total evaluation time across different numbers of robots, it can be noted that as the number of robots increases, the arena becomes more crowded and obstacle avoidance becomes harder, thus causing the average fitness to decrease. Also, performances are noisier (see larger error bars in lower right corner) and therefore there is less impact of a decreased N_p or N_i (flatter profile than with 1-2 robots). The following sections present a more detailed analysis of each individual parameter.

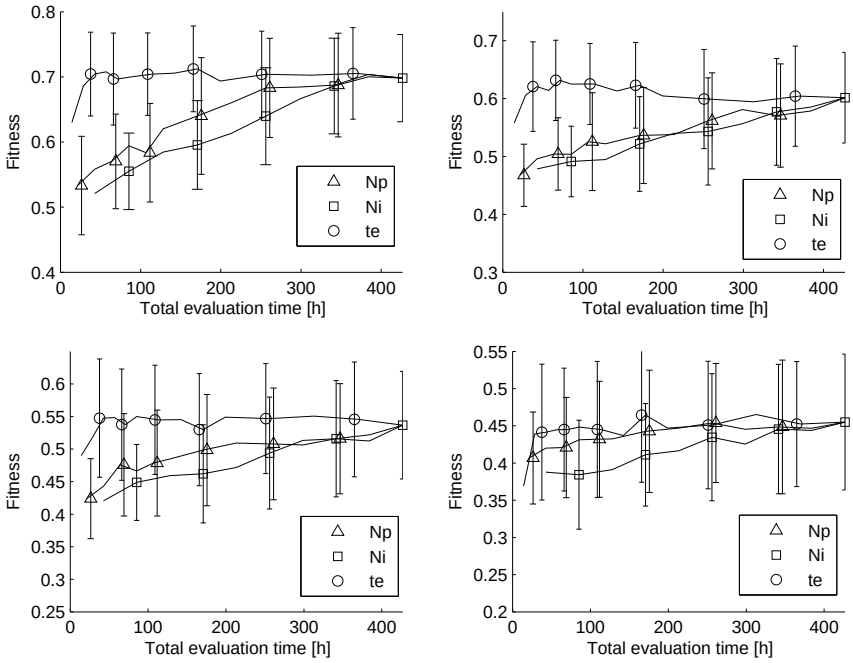

Fig. 1 Fitness as a function of total evaluation time for 1, 2, 4, and 8 robots respectively. Each curve represents the reduction of one individual parameter (population size, number of iterations, and evaluation span) with the others held constant.

4.2 Evaluation Span

To analyze the effect of the evaluation span, we reduced t_e from 150 s to 5 s, with 20 s steps in the higher range and 5 s steps in the low range. We did 100 runs for each value, and plotted the mean and standard deviation in Figure 2, left.

The mean fitness remains fairly constant for $30\,\text{s} < t_e < 150\,\text{s}$ for all number of robots. In fact, the difference in fitness between 150 s and 30 s is not statistically significant in all cases (Mann-Whitney U test, 5% significance level). For $t_e < 30\,\text{s}$ the fitness starts to decrease, although at different rates for different numbers of robots. In particular, for 8 robots t_e can be reduced to 10 s without a major change in fitness, which suggests that a crowded arena may allow for shorter evaluation spans due to more frequent collisions, and thus more opportunities to learn to avoid them. It is interesting to note that reducing the evaluation span does not seem to increase the fitness variation between runs.

The evaluation span parameter depends on the task and the environment. With the goal of trying to explain the lower limits for our task, we varied the evaluation span for several arena sizes using one robot (Figure 2, right). In this case, the point where performance starts to drop occurs at longer evaluation spans for larger arena

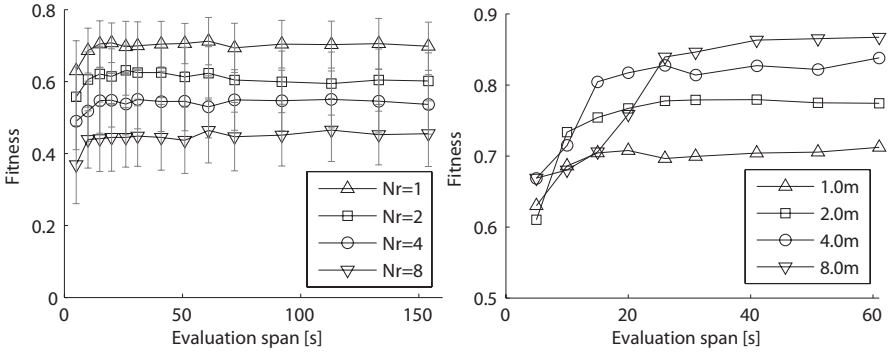

Fig. 2 Left: Mean fitness for different evaluation span values and number of robots in a 1x1 m arena. Error bars represent one standard deviation. Right: Mean fitness for different evaluation span values and arena sizes

sizes (15 and 25 seconds for 4 and 8 meters respectively). We suspect this point is related to the minimum time it takes to have at least one collision. In fact, if the robot moves in a straight line at a maximum speed of 0.25 m/s, it takes 16 and 32 seconds to cross one side of an arena of 4 and 8 meters respectively. Therefore, the robot speed and environment size can be used as guidelines to choose an evaluation span. We suggest that, in general, the minimum evaluation span should guarantee at least one interaction of the robot with other components of the environment relevant to the task at hand.

4.3 Re-evaluations

We then compared the performance of noise-resistant PSO with standard PSO to determine if re-evaluations improve performance in limited time scenarios. For any given set of parameters, noise-resistant PSO takes twice as much evaluation time as standard PSO due to the personal best re-evaluations. In order to perform a fair comparison, if we remove re-evaluations we need to double one of the other parameters to keep total evaluation time constant. We thus compared four alternatives: re-evaluations, doubled iterations, doubled evaluation span, and doubled population size. We performed 100 runs for each algorithmic variant and plotted the final fitness in Figure 3.

In the single robot case, noise-resistant PSO performed significantly worse than standard PSO with doubled iterations and doubled population size. However, as the number of robots is increased, the relative performance of noise-resistant PSO improves: for 2 robots there is no significant difference, and for 4 and 8 robots noise-resistant PSO significantly outperforms standard PSO with doubled iterations and doubled population size. It is worth noting that there is no significant difference between doubling population size and number of iterations for all number of robots,

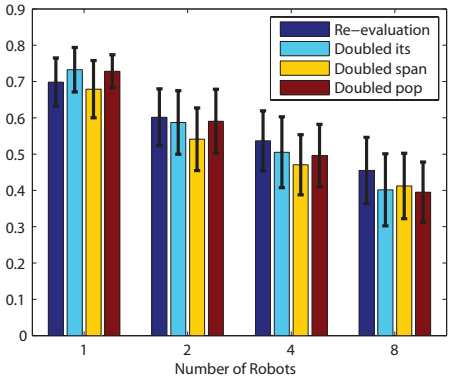

Fig. 3 Average fitness and standard deviation for noise-resistant PSO, standard PSO with doubled number of iterations, standard PSO with doubled evaluation span, and standard PSO with doubled population size.

and that doubling the evaluation span performs significantly worse in all cases except for 8 robots.

These results suggest that the decision to invest time in re-evaluations depends on the amount of uncertainty in fitness evaluations. In fact, as the arena becomes more crowded, there is more uncertainty in fitness evaluations, which depend both on the performance of other robots (avoiding other robots is easier if other robots are also trying to avoid you) and on initial conditions such as position in the arena (a robot is more likely to be trapped against a corner in a crowded arena).

Re-evaluations may also reduce the effect of heterogeneities on solution sharing in multi-robot evaluations, as shared solutions are re-evaluated at each iteration and thus can be dropped if they do not perform as well as they had done on other robots. The final advantage of re-evaluations can be seen in the case of dynamic environments, where a previously found good solution may no longer be valid after a certain amount of time. Thus, re-evaluations seem to be a good recommendation in general for multi-robot learning scenarios.

4.4 Population Size

Both for our robotic case study and the benchmark functions, the population size was reduced from 100 to 30 in steps of 10, and from 30 to 5 in steps of 5, in order to obtain more data points in what we expected to be an interesting region, while keeping all the other parameters the same as in the baseline. We did 100 independent runs for each value, results are shown in Figure 4, left and Figure 5.

It is clear from figures 1 and 5 that, at least with our baseline parameters, reducing the population size is better in terms of mean fitness than reducing the number of iterations, both for obstacle avoidance and for all benchmark functions[1]. Now the question that arises is how low should we set the population size? While there is

[1] Note that in benchmark functions lower fitness values mean better performance

no clear consensus in PSO literature [2], there are a few guidelines based on the dimension D of the search space such as $N_p = D$ or $N_p = 10 + 2\sqrt{D}$ [2].

Another approach is to start with a fixed value such as $N_p = 40$ and restart the algorithm with a larger N_p if early convergence is noticed. However, it is hard to determine if a restart is needed, especially when the maximum feasible fitness is not clear beforehand, which is often the case when learning robotic behaviors.

In Figure 4, left we note a slight change in the fitness slope at around $N_p = 25$, but this effect is much more clear in the case of the benchmark functions f_2, f_3, and f_4 (Figure 5, $N_p = 25$ and 500 default iterations, as mentioned in Table 1, corresponding to 12500 function evaluations).

Also, when population size becomes small, more outliers appear due to runs that fail to converge to a satisfactory solution. This can be noted in the higher standard deviation seen in reduced N_p as compared to reduced N_i with the same total evaluation time (see Figure 1).

Thus, because of higher fitness, lower variance, the possibility to distribute particles among robots, and the impracticality of the restart process, we prefer to err on the side of larger population sizes, and we suggest the following guideline:

$$N_p = \max(D, N_{rob}) \tag{6}$$

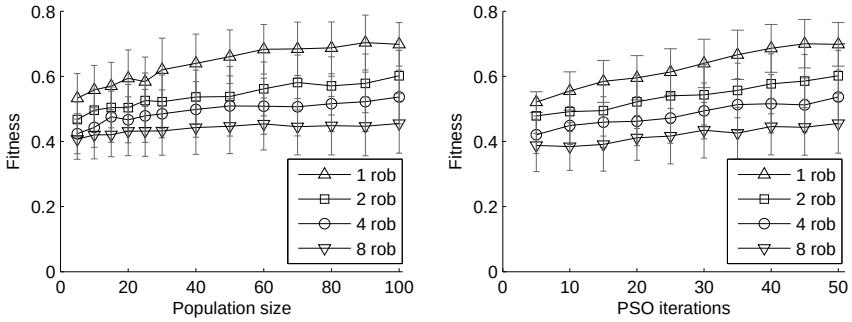

Fig. 4 Left: Mean fitness for different population size values. Right: Mean fitness for different number of iterations. Error bars represent one standard deviation.

4.5 PSO iterations

For our robotic case study, the number of iterations was reduced from 50 to 5 in steps of 5, again keeping all the other parameters the same as in the baseline. We

[2] This last formula is used in Standard PSO 2006, an effort to define a PSO standard published online in Particle Swarm Central `http://www.particleswarm.info`.

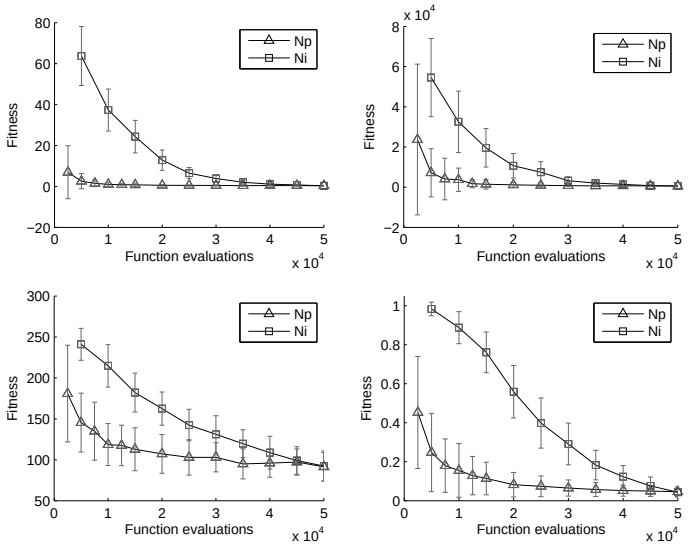

Fig. 5 Fitness as a function of number of evaluations for benchmark functions f1, f2, f3, and f4. Each curve represents the reduction of one individual parameter (population size and number of iterations) with the other held constant.

did 100 independent runs for each parameter value, results are shown in Figure 4, right. The chosen benchmark functions traditionally use larger values of N_i, so we chose 500 as a baseline and reduced it to 50 in steps of 50 (see Figure 5).

We observed a nearly linear performance drop for obstacle avoidance and on benchmark functions f_3 and f_4. For f_1 and f_2, the behavior of N_i was similar to that of N_p, but with a worse fitness overall.

Given that N_i is the easiest parameter to adjust on the fly, this parameter seems suitable for trade-offs between performance and available learning time. That is, for a fixed available time t_{wc}, we suggest using the previous guidelines to determine the 3 other parameters and allocate all remaining time to N_i using Eq. 7, which is derived from Eq. 4.

$$N_i = \frac{t_{wc}}{t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot (N_{re}+1)} \tag{7}$$

4.6 Limited Time Adaptation

For our limited time adaptation runs, we set a maximum total evaluation time of 8 h, which if distributed among 8 robots results in a wall-clock time of 1 h, about one

third of the battery autonomy of our robots[3]. Following our proposed guidelines, we used $N_p = 24$, $t_e = 20$ s, $N_{re} = 1$, and $N_i = 30$, and run the adaptation process in simulation for $N_{rob} = \{1, 2, 4, 8\}$ (see Figure 6).

The fitness difference between full-time and limited time adaptation is 17%, 17%, 14%, and 9% for 1, 2, 4, and 8 robots respectively. These values are relatively low considering the evaluation time was reduced more than 52 times. More importantly, both limited and full-time adaptation converged to the same obstacle avoidance strategy, regardless of the number of robots: going back and forth between walls in a straight line, reversing the direction of motion every time an obstacle was detected. We verified the sameness of the solution strategies by analyzing the trajectories described by the robots, focusing on the step length and angle distributions as described in [4].

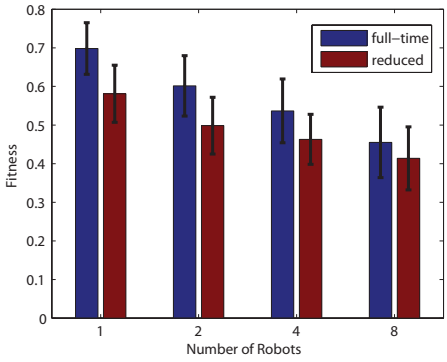

Fig. 6 Average fitness and standard deviation for full-time and limited-time adaptation.

5 Conclusion

We analyzed the effect of the four PSO algorithmic parameters that determine total evaluation time for a case study of multi-robot limited time learning of an obstacle avoidance behavior. Each parameter was varied independently, and based on the resulting fitness we proposed guidelines to choose these parameters for the case of multi-robot distributed learning. To add more generality to our guidelines, we ran analogous tests on benchmark functions traditionally used for numerical optimization problems.

For the population size parameter, we suggested to use at least the dimension of the search space D, and to use the number of robots if it is greater than D to take advantage of parallel evaluations and increased robustness. We proposed using the robot speed and environment size as guidelines to choose an evaluation span that

[3] Our autonomy is lower than that specified by the manufacturer due to additional modules such as an active tracking turret and a Wi-Fi card.

guarantees at least one interaction of the robot with other components of the environment relevant to the task at hand. Due to the inherent uncertainty in controller evaluations when using more than one robot, we showed that re-evaluations have a positive impact in multi-robot learning scenarios. The last parameter, the number of iterations, can be adjusted to fit the total evaluation time available.

By applying our guidelines, we were able to reduce the total adaptation time to an amount which can be easily completed without fully depleting the batteries of the individual robots. This resulted in a maximum quantitative performance drop of 17% but with no observable difference in the qualitative behaviors of the solutions. Even though we employed the PSO algorithm, we believe that the evaluation time issues and the guidelines presented in this paper are not limited to PSO and are relevant to population-based multi-robot learning in general.

Our next step is to validate these results with real robots. In particular, we intend to study the effect of asynchronous updates and dynamic neighborhood topologies on multi-robot learning. In the future, we hope to extend our analysis to tasks of increasing complexity, requiring a higher degree of coordination between robots. We are also interested in exploring PSO variations and other population-based algorithms that can be applied to limited-time distributed learning. Our final goal is to devise a set of general guidelines for fast, robust adaptation of high-performing robotic controllers.

Acknowledgements This research was supported by the Swiss National Science Foundation through the National Centre of Competence in Research Robotics.

References

1. Akat, S.B., Gazi, V.: Decentralized asynchronous particle swarm optimization. In: IEEE Swarm Intelligence Symposium (2008). DOI 10.1109/SIS.2008.4668304
2. Bratton, D., Kennedy, J.: Defining a Standard for Particle Swarm Optimization. In: IEEE Swarm Intelligence Symposium, pp. 120–127 (2007)
3. Chang, J., Chu, S., Roddick, J.: A parallel particle swarm optimization algorithm with communication strategies. Journal of Information Science pp. 809–818 (2005)
4. Di Mario, E., Mermoud, G., Mastrangeli, M., Martinoli, A.: A trajectory-based calibration method for stochastic motion models. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4341–4347 (2011)
5. Floreano, D., Mondada, F.: Evolution of homing navigation in a real mobile robot. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **26**(3), 396–407 (1996)
6. Hereford, J., Siebold, M.: Using the particle swarm optimization algorithm for robotic search applications. In: IEEE Swarm Intelligence Symposium, pp. 53–59 (2007)
7. Jin, Y., Branke, J.: Evolutionary Optimization in Uncertain EnvironmentsA Survey. IEEE Transactions on Evolutionary Computation **9**(3), 303–317 (2005)
8. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE International Conference on Neural Networks, pp. 1942 – 1948 vol.4 (1995)
9. Marques, L., Nunes, U., Almeida, A.T.: Particle swarm-based olfactory guided search. Autonomous Robots **20**(3), 277–287 (2006)
10. Michel, O.: Webots: Professional Mobile Robot Simulation. Advanced Robotic Systems **1**(1), 39–42 (2004)

11. Pan, H., Wang, L., Liu, B.: Particle swarm optimization for function optimization in noisy environment. Applied Mathematics and Computation **181**(2), 908–919 (2006)
12. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimizer in Noisy and Continuously Changing Environments. In: M.H. Hamza (ed.) Artificial Intelligence and Soft Computing, pp. 289–294. IASTED/ACTA Press (2001)
13. Poli, R.: Analysis of the publications on the applications of particle swarm optimisation. Journal of Artificial Evolution and Applications **2008**(2), 1–10 (2008)
14. Pugh, J., Martinoli, A.: Distributed scalable multi-robot learning using particle swarm optimization. Swarm Intelligence **3**(3), 203–222 (2009)
15. Pugh, J., Zhang, Y., Martinoli, A.: Particle swarm optimization for unsupervised robotic learning. In: IEEE Swarm Intelligence Symposium, pp. 92–99 (2005)
16. Rada-Vilela, J., Zhang, M., Seah, W.: Random Asynchronous PSO. The 5th International Conference on Automation, Robotics and Applications pp. 220–225 (2011)
17. Turduev, M., Atas, Y.: Cooperative Chemical Concentration Map Building Using Decentralized Asynchronous Particle Swarm Optimization Based Search by Mobile Robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4175–4180 (2010)